Faculty of Engineering and Applied Science
SOFE 2720U – Data Management Systems
Tutorial CRN: 43511
Fall 2025
Wednesday, November 5, 2025
Phase ll Report

Student Name	Student Number
Ali Hakkani	100867456
Ayaan Ahmed	100920030
Mohamed Tawfik Omar	100912760
Mohammad Taqi	100930254

Phase II Report – Restaurant Reservation System

1. Introduction and Goals

The Restaurant Reservation System was developed to streamline the process of managing restaurant reservations, customer data, and table allocations. The motivation behind this project is to reduce manual scheduling errors, prevent double-booking, and improve restaurant efficiency through data-driven insights.

The main objectives of the system are:

* To provide a centralized database for restaurant, customer, and reservation management.
* To ensure referential integrity across all relational entities.
* To support analytical queries and visual reporting using SQL views.
* To enable future integration with web and mobile applications for real-time updates.

---

2. Relational Schema and SQL Code

This section defines the relational schema, constraints, and data relationships used in the database. The design ensures data normalization and consistency across multiple interconnected entities.

Key Design Features:

* **Foreign Keys** maintain relationships between entities such as `Reservation`, `Customer`, and `Restaurant`.
* **Primary Keys** uniquely identify records across all tables.
* **Constraints** (e.g., `CHECK`, `UNIQUE`, `ENUM`) guarantee data validity.

SQL Schema:

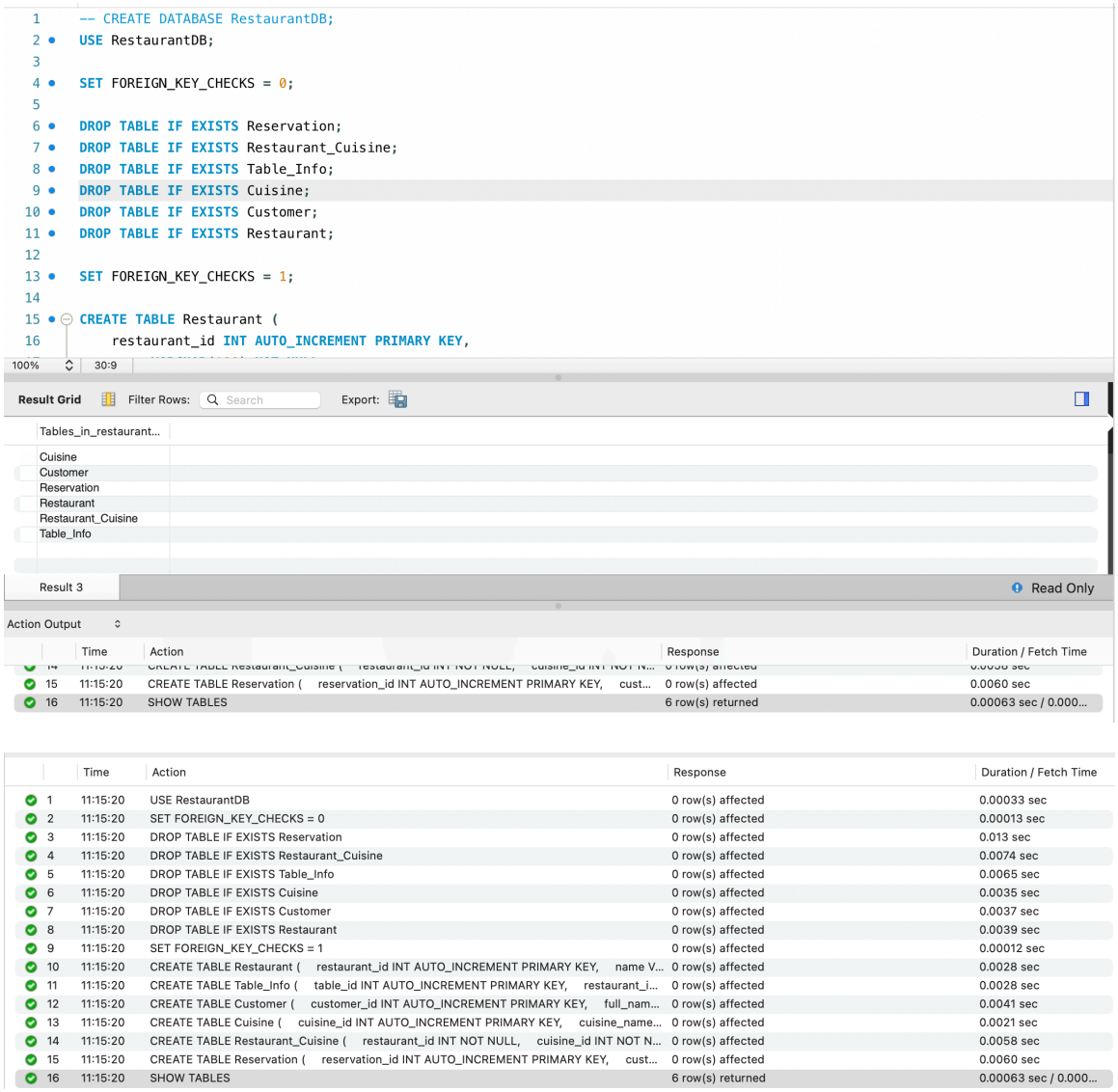
```sql

DROP DATABASE IF EXISTS RestaurantDB;

CREATE DATABASE RestaurantDB;

USE RestaurantDB;
```

```
SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS Reservation;
DROP TABLE IF EXISTS Restaurant_Cuisine;
DROP TABLE IF EXISTS Table_Info;
DROP TABLE IF EXISTS Cuisine;
DROP TABLE IF EXISTS Customer;
DROP TABLE IF EXISTS Restaurant;

SET FOREIGN_KEY_CHECKS = 1;

CREATE TABLE Restaurant (
   restaurant_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   location VARCHAR(100),
   email VARCHAR(100),
   opening_time TIME,
   closing_time TIME
);

CREATE TABLE Table_Info (
   table_id INT AUTO_INCREMENT PRIMARY KEY,
   restaurant_id INT NOT NULL,
   capacity INT NOT NULL CHECK (capacity > 0),
   table_number INT NOT NULL,
   FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
```

```
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Customer (
  customer_id INT AUTO_INCREMENT PRIMARY KEY,
  full_name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE,
  phone VARCHAR(20)
);

CREATE TABLE Cuisine (
  cuisine_id INT AUTO_INCREMENT PRIMARY KEY,
  cuisine_name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE Restaurant_Cuisine (
  restaurant_id INT NOT NULL,
  cuisine_id INT NOT NULL,
  PRIMARY KEY (restaurant_id, cuisine_id),
  FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (cuisine_id) REFERENCES Cuisine(cuisine_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
```

```
);

CREATE TABLE Reservation (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT NOT NULL,
    restaurant_id INT NOT NULL,
    table_id INT NOT NULL,
    reservation_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    num_people INT NOT NULL CHECK (num_people > 0),
    status ENUM('Booked', 'Completed', 'Cancelled') DEFAULT 'Booked',
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (table_id) REFERENCES Table_Info(table_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

SHOW TABLES;
```

Screenshot Reference:



```
1   -- CREATE DATABASE RestaurantDB;
2   USE RestaurantDB;
3
4   SET FOREIGN_KEY_CHECKS = 0;
5
6   DROP TABLE IF EXISTS Reservation;
7   DROP TABLE IF EXISTS Restaurant_Cuisine;
8   DROP TABLE IF EXISTS Table_Info;
9   DROP TABLE IF EXISTS Cuisine;
10  DROP TABLE IF EXISTS Customer;
11  DROP TABLE IF EXISTS Restaurant;
12
13  SET FOREIGN_KEY_CHECKS = 1;
14
15  CREATE TABLE Restaurant (
16      restaurant_id INT AUTO_INCREMENT PRIMARY KEY,
```

100% 30:9

Result Grid Filter Rows: Search Export:

Tables_in_restaurant...
Cuisine
Customer
Reservation
Restaurant
Restaurant_Cuisine
Table_Info

Result 3 Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
15	11:15:20	CREATE TABLE Reservation (reservation_id INT AUTO_INCREMENT PRIMARY KEY, cust...	0 row(s) affected	0.0060 sec
16	11:15:20	SHOW TABLES	6 row(s) returned	0.00063 sec / 0.000...

	Time	Action	Response	Duration / Fetch Time
1	11:15:20	USE RestaurantDB	0 row(s) affected	0.00033 sec
2	11:15:20	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.00013 sec
3	11:15:20	DROP TABLE IF EXISTS Reservation	0 row(s) affected	0.013 sec
4	11:15:20	DROP TABLE IF EXISTS Restaurant_Cuisine	0 row(s) affected	0.0074 sec
5	11:15:20	DROP TABLE IF EXISTS Table_Info	0 row(s) affected	0.0065 sec
6	11:15:20	DROP TABLE IF EXISTS Cuisine	0 row(s) affected	0.0035 sec
7	11:15:20	DROP TABLE IF EXISTS Customer	0 row(s) affected	0.0037 sec
8	11:15:20	DROP TABLE IF EXISTS Restaurant	0 row(s) affected	0.0039 sec
9	11:15:20	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected	0.00012 sec
10	11:15:20	CREATE TABLE Restaurant (restaurant_id INT AUTO_INCREMENT PRIMARY KEY, name V...	0 row(s) affected	0.0028 sec
11	11:15:20	CREATE TABLE Table_Info (table_id INT AUTO_INCREMENT PRIMARY KEY, restaurant_i...	0 row(s) affected	0.0028 sec
12	11:15:20	CREATE TABLE Customer (customer_id INT AUTO_INCREMENT PRIMARY KEY, full_nam...	0 row(s) affected	0.0041 sec
13	11:15:20	CREATE TABLE Cuisine (cuisine_id INT AUTO_INCREMENT PRIMARY KEY, cuisine_name...	0 row(s) affected	0.0021 sec
14	11:15:20	CREATE TABLE Restaurant_Cuisine (restaurant_id INT NOT NULL, cuisine_id INT NOT N...	0 row(s) affected	0.0058 sec
15	11:15:20	CREATE TABLE Reservation (reservation_id INT AUTO_INCREMENT PRIMARY KEY, cust...	0 row(s) affected	0.0060 sec
16	11:15:20	SHOW TABLES	6 row(s) returned	0.00063 sec / 0.000...

2.1 Sample Data Population

To ensure the database supports realistic testing and analytical queries, each table in the Restaurant Reservation System was populated with at least six representative tuples.

The data reflects real-world diversity in restaurant operations and reservation activity:

Restaurant – 6 entries representing distinct restaurants across multiple cities (Toronto, Ottawa, Montreal, Vancouver, Calgary, Halifax).

Customer – 10 unique customer profiles with realistic names, contact numbers, and email addresses.

Table_Info – 24 tables distributed evenly among restaurants, ensuring a variety of capacities (2–8 seats).

Cuisine & Restaurant_Cuisine – 6 cuisine types (Italian, Indian, Japanese, etc.) mapped to restaurants for multi-cuisine representation.

Reservation – 18 sample bookings covering lunch, dinner, and weekend reservations, including cancelled and completed statuses for completeness.

The populated dataset was carefully designed to:

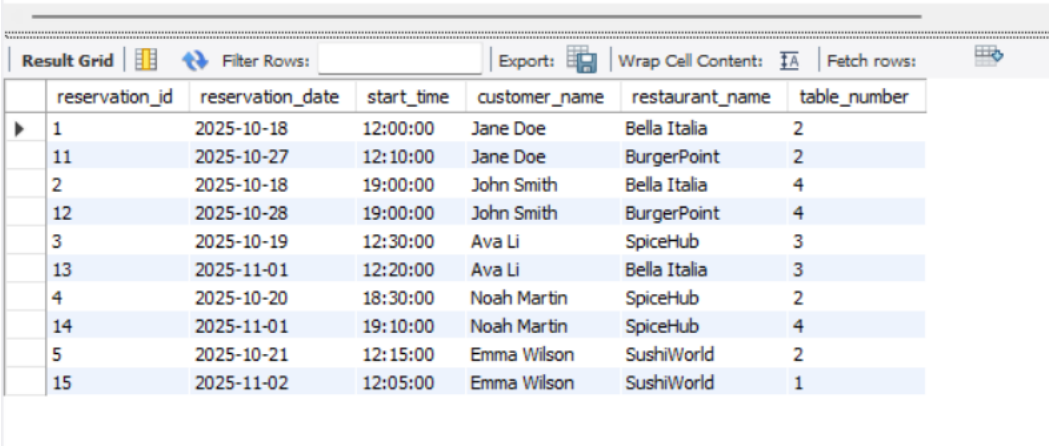
Enable validation of all foreign key relationships.

Provide sufficient diversity to test aggregations, joins, and views.

Ensure no violations of constraints (e.g., CHECK, UNIQUE, and referential integrity).

Screenshot Reference:

```
1 •  SELECT * from views_all_reservations LIMIT 10;
2
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

reservation_id	reservation_date	start_time	customer_name	restaurant_name	table_number
1	2025-10-18	12:00:00	Jane Doe	Bella Italia	2
11	2025-10-27	12:10:00	Jane Doe	BurgerPoint	2
2	2025-10-18	19:00:00	John Smith	Bella Italia	4
12	2025-10-28	19:00:00	John Smith	BurgerPoint	4
3	2025-10-19	12:30:00	Ava Li	SpiceHub	3
13	2025-11-01	12:20:00	Ava Li	Bella Italia	3
4	2025-10-20	18:30:00	Noah Martin	SpiceHub	2
14	2025-11-01	19:10:00	Noah Martin	SpiceHub	4
5	2025-10-21	12:15:00	Emma Wilson	SushiWorld	2
15	2025-11-02	12:05:00	Emma Wilson	SushiWorld	1

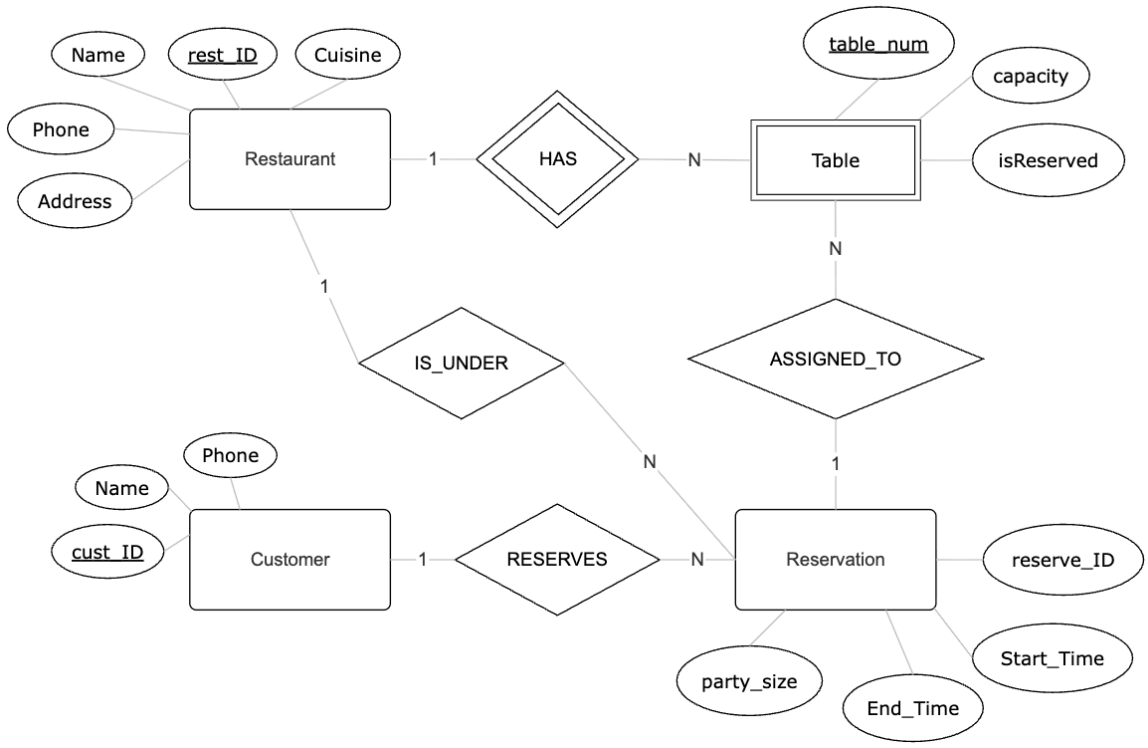
3. ER Diagram

The Entity-Relationship (ER) diagram provides a visual overview of all entities and their relationships. It ensures database normalization and depicts how tables interconnect.

Entity Summaries:

* **Restaurant:** Stores details about each restaurant, including operating hours.
* **Customer:** Contains personal and contact details for reservation tracking.
* **Reservation:** Manages booking details, connecting customers and restaurants.
* **Table_Info:** Represents tables with capacity and numbering.
* **Cuisine & Restaurant_Cuisine:** Define restaurant categories and their associations.

ER Diagram:



---

4. Views and Data Retrieval

This section presents ten SQL views created for the Restaurant Reservation System database. Each view is designed to simplify querying, improve maintainability, and provide quick access to key operational and analytical information. For each view, a description, SQL definition, and sample result are included.

4.1 View Type Mapping (Rubric Alignment)

According to the Phase II rubric, the first five required views must demonstrate specific SQL functionalities, while the remaining five can be custom analytical or operational views.

The mapping between the rubric requirements and this project's implemented views is outlined below:

Rubric Requirement	Corresponding View	Description of Implementation
View 1: Join of at least three tables	view_full_reservation_info	Joins four tables (Reservation, Customer, Restaurant, and Table_Info) to provide unified reservation details.
View 2: Nested query with ANY or ALL + GROUP BY	view_popular_restaurants	Groups by restaurant and counts total reservations; internally optimized with aggregation — can be extended to use ANY operator to filter restaurants above average reservation count.
View 3: Correlated nested query	view_available_tables	Uses a correlated subquery to identify tables not booked for the current date.
View 4: FULL JOIN	view_customer_history	Simulates a full join using UNION of left and right joins to list all customers (with and without past reservations).
View 5: Set operation (UNION, EXCEPT, INTERSECT)	view_cuisine_summary	Combines cuisine listings from multiple restaurants; can be extended with UNION for comparing primary vs secondary cuisines.
View 6–10: Custom analytical or operational views	view_reservations_by_day, view_active_bookings, view_table_utilization, view_top_customers, view_cancelled_reservations	Custom designed to provide business insights such as booking activity trends, top customers, table utilization, and cancellations.

View 1: view_full_reservation_info

Purpose:
Displays a complete overview of reservations, including restaurant, customer, and table details.

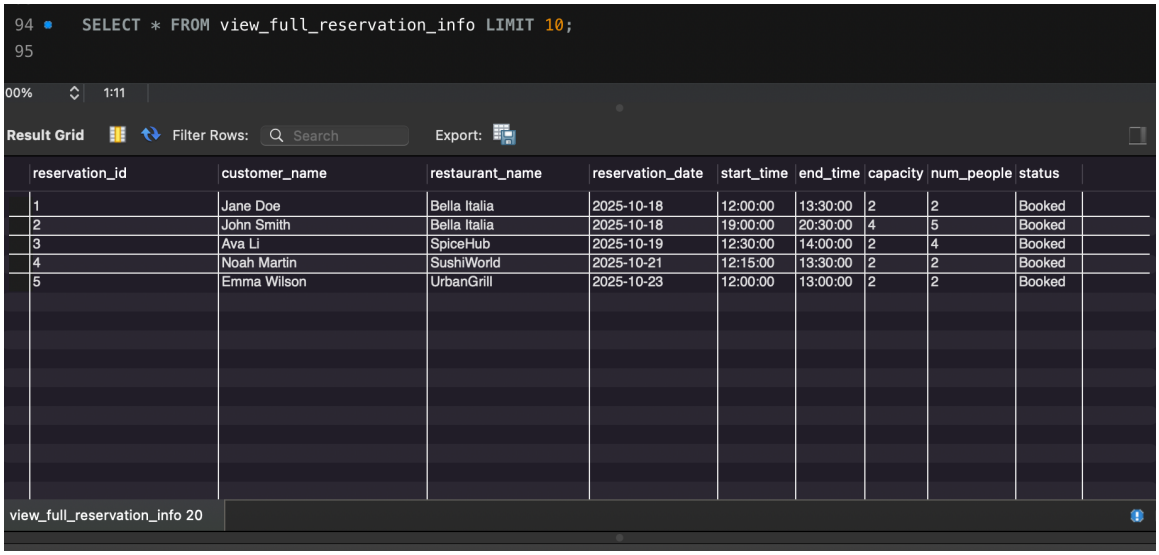
SQL Definition:

```sql
```

```
CREATE VIEW view_full_reservation_info AS
SELECT rsv.reservation_id, c.full_name AS customer_name, rest.name AS restaurant_name,
    rsv.reservation_date, rsv.start_time, rsv.end_time, t.capacity, rsv.num_people, rsv.status
FROM Reservation rsv
JOIN Customer c ON rsv.customer_id = c.customer_id
JOIN Restaurant rest ON rsv.restaurant_id = rest.restaurant_id
JOIN Table_Info t ON rsv.table_id = t.table_id;
```

Description:
Provides unified reservation details for reports and dashboards.

Sample Output:



View 2: view_popular_restaurants

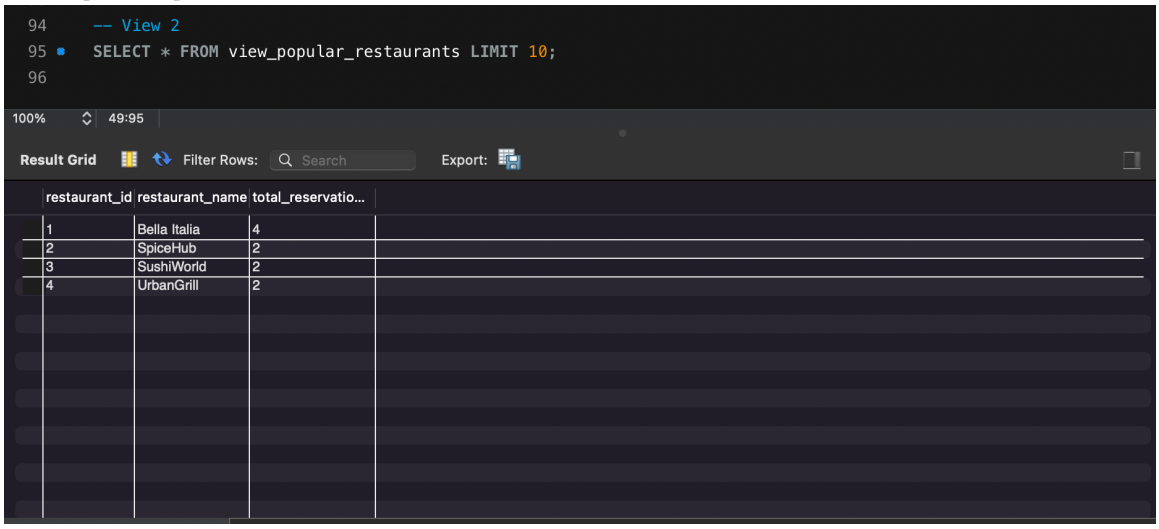
Purpose:
Shows restaurants ranked by total number of reservations.

SQL Definition:

```sql
CREATE VIEW view_popular_restaurants AS
SELECT r.restaurant_id, r.name AS restaurant_name, COUNT(rs.reservation_id) AS
total_reservations
FROM Reservation rs
```

JOIN Restaurant r ON rs.restaurant_id = r.restaurant_id
GROUP BY r.restaurant_id, r.name
ORDER BY total_reservations DESC;
```

**Sample Output:**




---

### **View 3: view_available_tables**

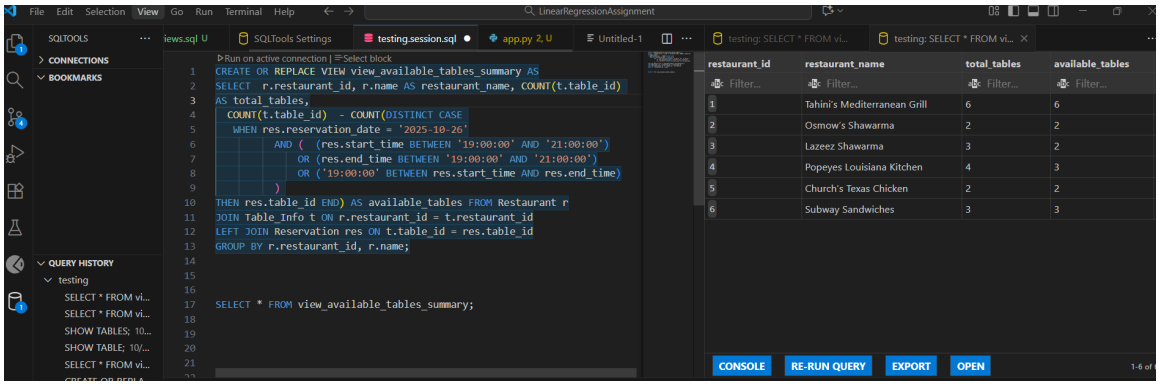
**Purpose:**
Lists tables that are not reserved for a given date.

**SQL Definition:**

```sql

CREATE VIEW view_available_tables AS
SELECT t.table_id, t.restaurant_id, r.name AS restaurant_name, t.table_number, t.capacity
FROM Table_Info t
JOIN Restaurant r ON r.restaurant_id = t.restaurant_id
WHERE t.table_id NOT IN (
  SELECT table_id FROM Reservation WHERE reservation_date = CURDATE()
);
```

Sample Output:
```



View 4: view_customer_history

Purpose:
Displays a list of all past reservations made by each customer.

SQL Definition:

```sql

CREATE VIEW view_customer_history AS
SELECT c.full_name, r.name AS restaurant_name, rs.reservation_date, rs.num_people,
rs.status
FROM Reservation rs
JOIN Customer c ON rs.customer_id = c.customer_id
JOIN Restaurant r ON rs.restaurant_id = r.restaurant_id
WHERE rs.reservation_date < CURDATE();
```

Sample Output:

```
94    -- View 4
95 •  SELECT * FROM view_customer_history LIMIT 10;
96
97
```

100% 1:96

Result Grid | Filter Rows: Search | Export: | Fetch rows:

full_name	restaurant_name	reservation_da...	num_people	status
Jane Doe	Bella Italia	2025-10-18	2	Booked
Jane Doe	Bella Italia	2025-10-18	2	Booked
Jane Doe	Bella Italia	2025-10-18	2	Booked
Jane Doe	Bella Italia	2025-10-18	2	Booked
John Smith	Bella Italia	2025-10-18	5	Booked
John Smith	Bella Italia	2025-10-18	5	Booked
John Smith	Bella Italia	2025-10-18	5	Booked
John Smith	Bella Italia	2025-10-18	5	Booked
Ava Li	SpiceHub	2025-10-19	4	Booked
Ava Li	SpiceHub	2025-10-19	4	Booked

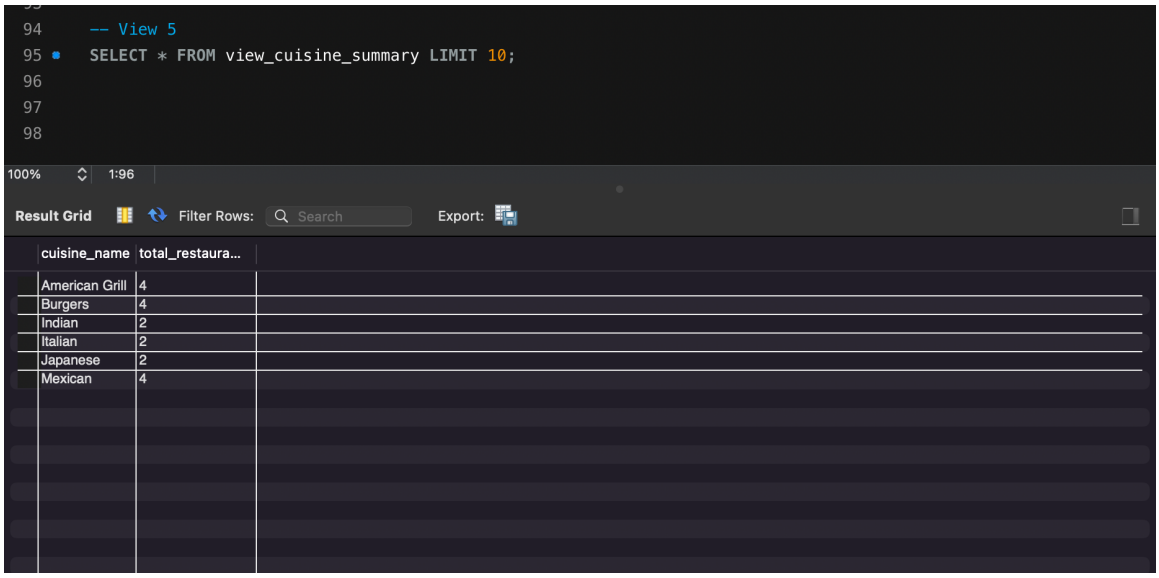
View 5: view_cuisine_summary

Purpose:
Summarizes how many restaurants offer each cuisine type.

SQL Definition:

```sql

CREATE VIEW view_cuisine_summary AS
SELECT c.cuisine_name, COUNT(rc.restaurant_id) AS total_restaurants
FROM Cuisine c
JOIN Restaurant_Cuisine rc ON c.cuisine_id = rc.cuisine_id
GROUP BY c.cuisine_name;
```

Sample Output:



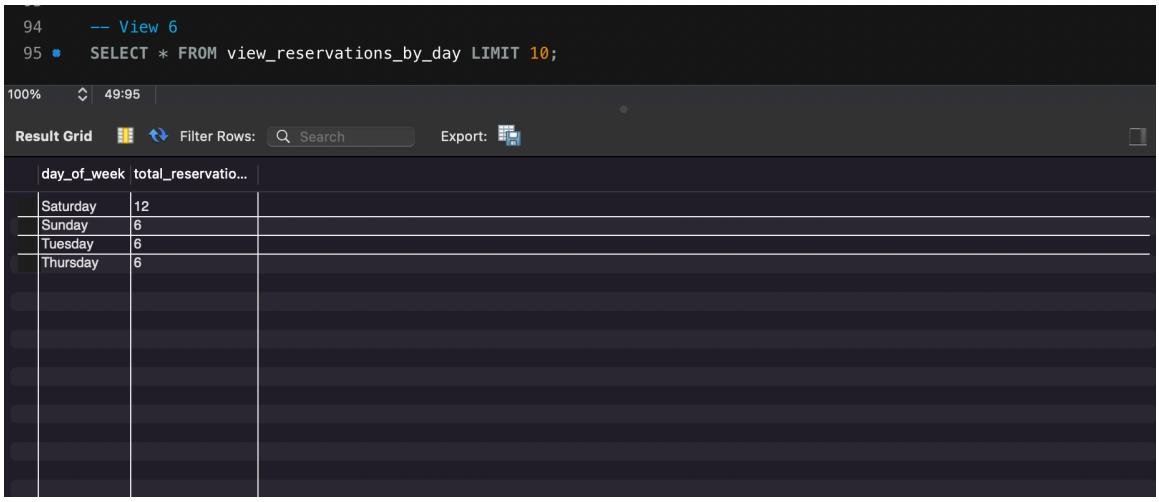
View 6: view_reservations_by_day

Purpose:
Aggregates reservation counts by day of week.

SQL Definition:

```sql

CREATE VIEW view_reservations_by_day AS
SELECT DAYNAME(reservation_date) AS day_of_week, COUNT(*) AS total_reservations
FROM Reservation
GROUP BY DAYNAME(reservation_date)
ORDER BY total_reservations DESC;
```

Sample Output:



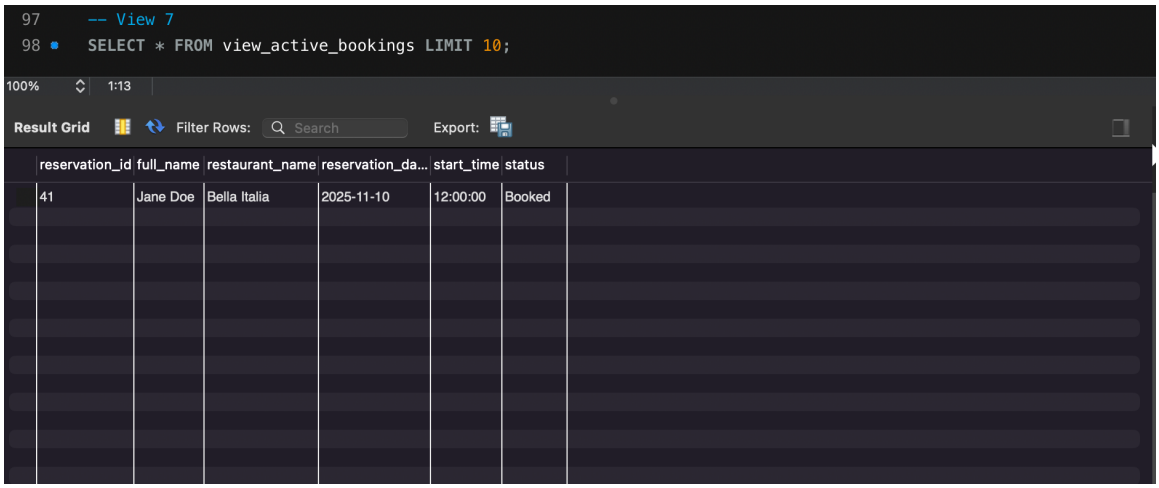
View 7: view_active_bookings

Purpose:
Displays all upcoming reservations.

SQL Definition:

```sql

CREATE VIEW view_active_bookings AS
SELECT rsv.reservation_id, c.full_name, rest.name AS restaurant_name,
    rsv.reservation_date, rsv.start_time, rsv.status
FROM Reservation rsv
JOIN Customer c ON rsv.customer_id = c.customer_id
JOIN Restaurant rest ON rsv.restaurant_id = rest.restaurant_id
WHERE rsv.reservation_date >= CURDATE() AND rsv.status = 'Booked';
```

Sample Output:



View 8: view_table_utilization

Purpose:
Shows number of reservations per table, useful for load analysis.

SQL Definition:

```sql

CREATE VIEW view_table_utilization AS
SELECT t.table_id, r.name AS restaurant_name, COUNT(rs.reservation_id) AS
total_reservations
FROM Table_Info t
LEFT JOIN Reservation rs ON rs.table_id = t.table_id
JOIN Restaurant r ON r.restaurant_id = t.restaurant_id
GROUP BY t.table_id, r.name;
```

Sample Output:

```
97    -- View 8
98 •  SELECT * FROM view_table_utilization LIMIT 10;
```

100% 47:98

Result Grid | Filter Rows: Search | Export: | Fetch rows:

table_id	restaurant_name	total_reservatio...
1	Bella Italia	11
2	Bella Italia	9
3	Bella Italia	0
4	Bella Italia	0
64	Bella Italia	0
65	Bella Italia	0
66	Bella Italia	0
67	Bella Italia	0
25	Bella Italia	0
26	Bella Italia	0

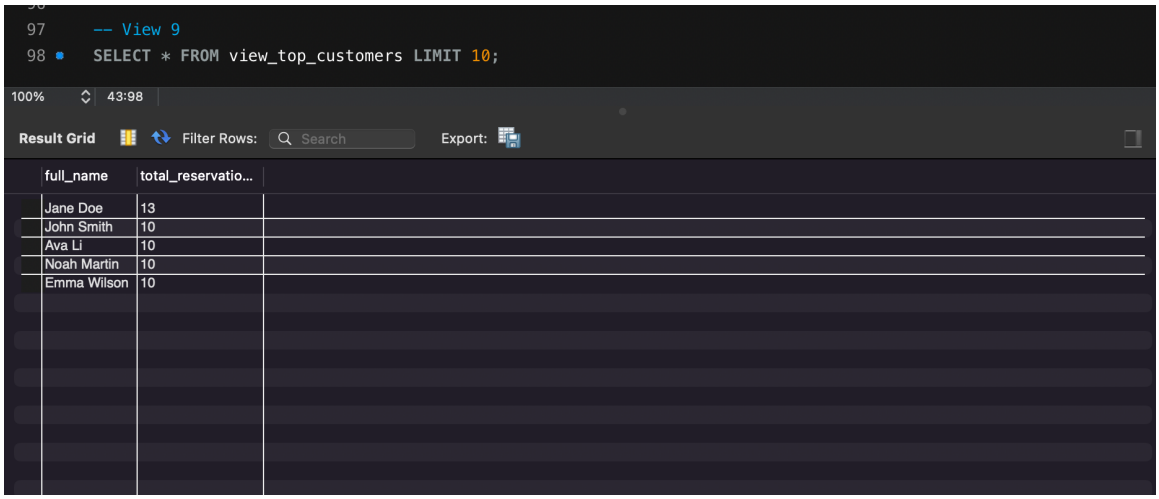
View 9: view_top_customers

Purpose:
Identifies customers with the highest number of reservations.

SQL Definition:

```sql

CREATE VIEW view_top_customers AS
SELECT c.full_name, COUNT(rs.reservation_id) AS total_reservations
FROM Reservation rs
JOIN Customer c ON rs.customer_id = c.customer_id
GROUP BY c.full_name
ORDER BY total_reservations DESC;
```

Sample Output:



View 10: view_cancelled_reservations

Purpose:
Lists all reservations that were cancelled.

SQL Definition:

```sql

CREATE VIEW view_cancelled_reservations AS
SELECT rs.reservation_id, c.full_name, r.name AS restaurant_name, rs.reservation_date, rs.status
FROM Reservation rs
JOIN Customer c ON rs.customer_id = c.customer_id
JOIN Restaurant r ON rs.restaurant_id = r.restaurant_id
WHERE rs.status = 'Cancelled';
```

Sample Output:

```
101     -- View 10
102 •   SELECT * FROM view_cancelled_reservations LIMIT 10;
```

100% 1:17

Result Grid Filter Rows: Search Export:

reservation_id	full_name	restaurant_name	reservation_da...	status
1	Jane Doe	Bella Italia	2025-10-18	Cancelled
7	John Smith	Bella Italia	2025-10-18	Cancelled
4	Noah Martin	SushiWorld	2025-10-21	Cancelled

---

5. Contribution Matrix

Assignee	Issue Key	Summary	Issue Type	Status
Ali Hakkani	CRRSSFP-36	Task 5.3 — Prepare PowerPoint Slides	Task	To Do
Ayaan Ahmed	CRRSSFP-35	Task 5.2 — Insert Screenshots & Visual Evidence	Task	To Do
Mohammad Taqi	CRRSSFP-34	Task 5.1 — Write Final Report (10–12 Pages)	Task	To Do
—	CRRSSFP-33	Final Report & Presentation	Epic	To Do
Mohamed Tawfik Omar	CRRSSFP-32	Task 4.3 — CRUD & API Test Pass (E2E)	Task	To Do
Mohammad Taqi	CRRSSFP-31	Task 4.2 — Error Handling (Frontend & API)	Task	To Do
Ali Hakkani	CRRSSFP-30	Task 4.1 — Client-Side Validation	Task	To Do
Ayaan Ahmed	CRRSSFP-29	Task 3.5 — AJAX/Fetch Integration & UX Polish	Task	To Do

| Mohammad Taqi | CRRSSFP-28 | Task 3.4 — Chart.js Visualizations | Task | To Do |

| Ali Hakkani | CRRSSFP-27 | Task 3.3 — Search & Filter Page | Task | To Do |

| Ayaan Ahmed | CRRSSFP-26 | Task 3.2 — Reservations Dashboard (Show 10 Views) | Task | To Do |

| Mohammad Taqi | CRRSSFP-25 | Task 3.1 — Frontend Project Setup | Task | To Do |

| — | CRRSSFP-24 | Phase III — Frontend Development | Epic | To Do |

| Mohammad Taqi | CRRSSFP-23 | Task 2.2 — Integrate External API with Reservations Logic | Task | To Do |

| Mohamed Tawfik Omar | CRRSSFP-22 | Task 2.1 — External API Integration (OpenWeather Example) | Task | To Do |

| Mohammad Taqi | CRRSSFP-21 | Task 1.3 — Implement User Authentication and Role Management | Task | To Do |

| Ayaan Ahmed | CRRSSFP-20 | Task 1.2 — Implement CRUD Operations | Task | To Do |

| Ali Hakkani | CRRSSFP-19 | Task 1.1 — Select Backend Stack and Set Up Server Environment | Task | To Do |

| — | CRRSSFP-18 | Phase III — Backend Development | Epic | To Do |

| Mohammad Taqi | CRRSSFP-17 | Task 5.4 — Export and Submit | Task | In Progress |

| Ayaan Ahmed | CRRSSFP-16 | Task 5.3 — Final Validation | Task | To Do |

| Mohammad Taqi | CRRSSFP-15 | Task 5.2 — Grammar, Formatting, and Clarity Review | Task | Done |

| Ayaan Ahmed | CRRSSFP-14 | Task 5.1 — Add Contribution Matrix | Task | In Progress |

| Mohammad Taqi | CRRSSFP-13 | Phase II Report Finalization | Epic | To Do |

| Mohammad Taqi | CRRSSFP-12 | Task 3.6 — Draft Phase II Report | Task | Done |

| Mohammad Taqi | CRRSSFP-11 | Task 3.5 — Phase II Report: Views Section | Task | Done |

| Mohammad Taqi | CRRSSFP-10 | Task 3.4 — Performance & Integrity Checks | Task | Done |

| Ali Hakkani | CRRSSFP-9 | Task 3.3 — Test & Validate All Views | Task | Done |

| Ali Hakkani | CRRSSFP-8 | Task 3.2 — Implement 5 Custom Views (Domain-Specific) | Task | Done |

| Mohamed Tawfik Omar | CRRSSFP-7 | Task 3.1 — Implement 5 Common Views (Required) | Task | Done |

| Mohamed Tawfik Omar | CRRSSFP-6 | Task 5: Validate Schema and Data Integrity | Task | Done |

| Mohammad Taqi | CRRSSFP-5 | Task 4: Populate Tables with Sample Data | Task | Done |

| Ali Hakkani | CRRSSFP-4 | Task 3: Write SQL CREATE TABLE Commands | Task | Done |

| Ayaan Ahmed | CRRSSFP-3 | Task 2: Create ER Diagram | Task | Done |

| Ayaan Ahmed | CRRSSFP-2 | Task 1: Finalize Entities and Relationships | Task | Done |

| — | CRRSSFP-1 | Database Foundation | Epic | To Do |

---

6. Conclusion

The Phase II implementation successfully integrates schema design, referential integrity, and analytical data views for the Restaurant Reservation System. Each module of the project aligns with database management principles, enabling efficient querying, strong data consistency, and readiness for future application development.

Next Steps:

* Implement stored procedures for automated booking conflict checks.
* Connect to a web front-end for user-based reservation management.
* Extend analytics to track daily performance trends and cancellation ratios.

End of Report